**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**СВЯЗИ И ИНФОРМАТИКИ**

Кафедра «Информационная безопасность»

**ЛАБОРАТОРНАЯ РАБОТА**

**№6**

по дисциплине «Объектно-ориентированное программирование систем защиты информации»

на тему:

**«Классы с динамическими структурами данных. Шаблонные классы»**

Вариант №10

Выполнил: студ. гр. БПЗ1501

Лисеенко Я.А.

Проверил: ассистент кафедры ИБ

Барков В.В.

Москва 2018 г

# Цель работы

Изучить динамические структуры данных, овладеть навыками создания конструкторов

копирования, перемещения, деструкторов, перегрузки операций копирования и перемещения.

# Результаты

## Queue.h

#pragma once

#include "iostream"

#include "conio.h"

using namespace std;

class Queue

{

int \*arr;

int count;

int end = 0, start = 0, size = 0;

public:

Queue(int count); //инициализация

Queue(const Queue &other); //копирование

Queue(Queue &&other); //перемещение

Queue &operator=(const Queue &other); //присваивание

Queue &operator=(Queue &&other); //присваивание с еремещением

~Queue(); //деструктор

int GetSize();

void Push(int element); //Вставка (добавление) элемента

int Pop(); //Удаление (взятие) элемента

int Peek(); //Просмотр (взятие без удаления) элемента

bool CheckNoEmptyQueue(); //Проверка наличия элементов

friend ostream & operator << (ostream & stream, const Queue &a); //Вывод

};

## Queue.cpp

#include "Queue.h"

Queue::Queue(int count)

: count(count)

{

arr = new int[count];

}

Queue::~Queue()

{

delete[] arr;

}

Queue::Queue(const Queue &other)

{

arr = new int[other.count];

count = other.count;

end = other.end;

start = other.start;

size = other.size;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

Queue &Queue::operator=(const Queue &other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = new int[other.count];

count = other.count;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

Queue &Queue::operator=(Queue &&other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

Queue::Queue(Queue &&other)

{

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

int Queue::GetSize()

{

return this->size;

}

void Queue::Push(int element)

{

if (size < count)

{

arr[end] = element;

end = (end + 1) % count;

size++;

}

}

int Queue::Pop()

{

if (size > 0)

{

int element = arr[start];

start = (start + 1) % count;

size--;

return element;

}

return 0;

}

int Queue::Peek()

{

return arr[start];

}

bool Queue::CheckNoEmptyQueue()

{

return (size > 0);

}

ostream & operator << (ostream & stream, const Queue &a)

{

Queue b = a;

stream << "=====\n";

for (int i = 0; i < b.count; i++)

{

stream << "| " << b.Pop() << "\n";

}

stream << "=====\n";

return stream;

}

## QueueTemplate.h

#pragma once

#include "iostream"

#include "conio.h"

using namespace std;

template<typename T>

class QueueTemplate

{

T \*arr;

int count;

int end = 0, start = 0, size = 0;

public:

QueueTemplate(int count); //инициализация

QueueTemplate(const QueueTemplate &other); //копирование

QueueTemplate(QueueTemplate &&other); //перемещение

QueueTemplate &operator=(const QueueTemplate &other); //присваивание

QueueTemplate &operator=(QueueTemplate &&other); //присваивание с еремещением

~QueueTemplate(); //деструктор

int GetSize();

void Push(T element); //Вставка (добавление) элемента

T Pop(); //Удаление (взятие) элемента

T Peek(); //Просмотр (взятие без удаления) элемента

bool CheckNoEmptyQueue(); //Проверка наличия элементов

friend ostream & operator << <T>(ostream & stream, const QueueTemplate<T> &a); //Вывод

};

template<typename T>

QueueTemplate<T>::QueueTemplate(int count)

: count(count)

{

arr = new T[count];

}

template<typename T>

QueueTemplate<T>::~QueueTemplate()

{

delete[] arr;

}

template<typename T>

QueueTemplate<T>::QueueTemplate(const QueueTemplate &other)

{

arr = new T[other.count];

count = other.count;

end = other.end;

start = other.start;

size = other.size;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

template<typename T>

QueueTemplate<T> &QueueTemplate<T>::operator=(const QueueTemplate &other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = new T[other.count];

count = other.count;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

template<typename T>

QueueTemplate<T> &QueueTemplate<T>::operator=(QueueTemplate &&other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

template<typename T>

QueueTemplate<T>::QueueTemplate(QueueTemplate &&other)

{

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

template<typename T>

int QueueTemplate<T>::GetSize()

{

return this->size;

}

template<typename T>

void QueueTemplate<T>::Push(T element)

{

if (size < count)

{

arr[end] = element;

end = (end + 1) % count;

size++;

}

}

template<typename T>

T QueueTemplate<T>::Pop()

{

if (size > 0)

{

T element = arr[start];

start = (start + 1) % count;

size--;

return element;

}

return 0;

}

template<typename T>

T QueueTemplate<T>::Peek()

{

return arr[start];

}

template<typename T>

bool QueueTemplate<T>::CheckNoEmptyQueue()

{

return (size > 0);

}

template<typename T>

ostream & operator << (ostream & stream, const QueueTemplate<T> &a)

{

QueueTemplate<T> b = a;

stream << "=====\n";

for (int i = 0; i < b.count; i++)

{

stream << "| " << b.Pop() << "\n";

}

stream << "=====\n";

return stream;

}